# INTER IIT Tech Meet II.0

## The Robotic Charging Challenge

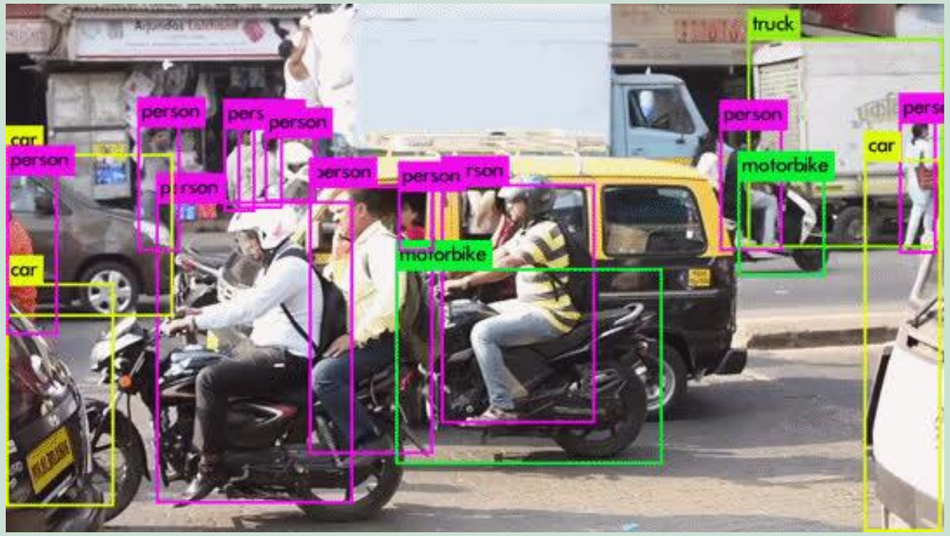
### An innovative attempt to boost performance


# Computer Vision Aid

## Charging Plug Location Detection

**Why Computer Vision?**

- With the current advent of Machine Learning, Object Detection has become really easy with the help of State of the art Models.
- Cameras have become relatively cheap and work as universal sensors.
- Computer Vision and deep learning algorithms provide high accuracy and can be made robust to different lighting and environment conditions.

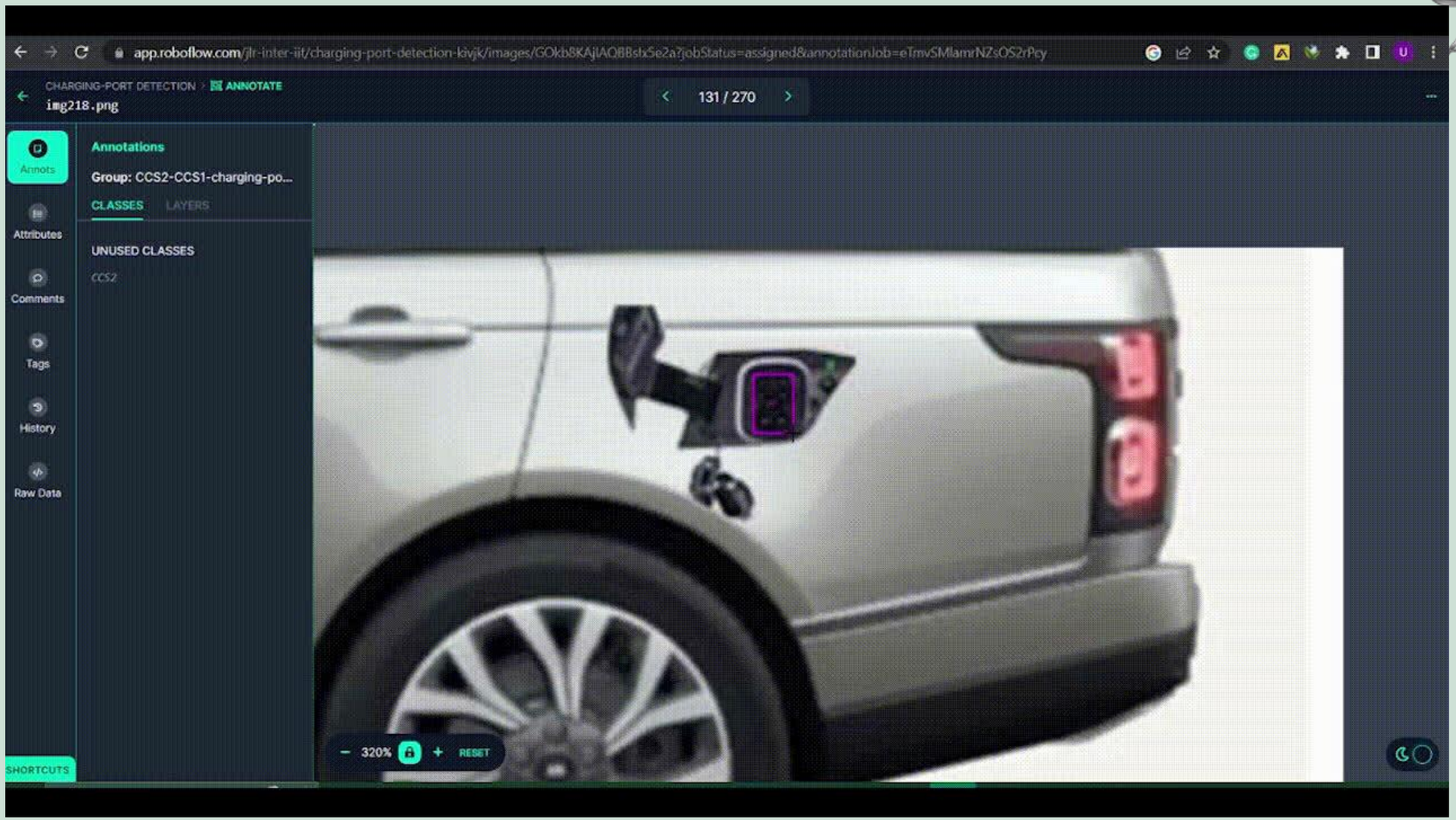


# Object Detection | YOLO v7

We used YOLO v7: the fastest and most accurate real-time object detection model for detecting CCS1 and CCS2 5 charging plugs

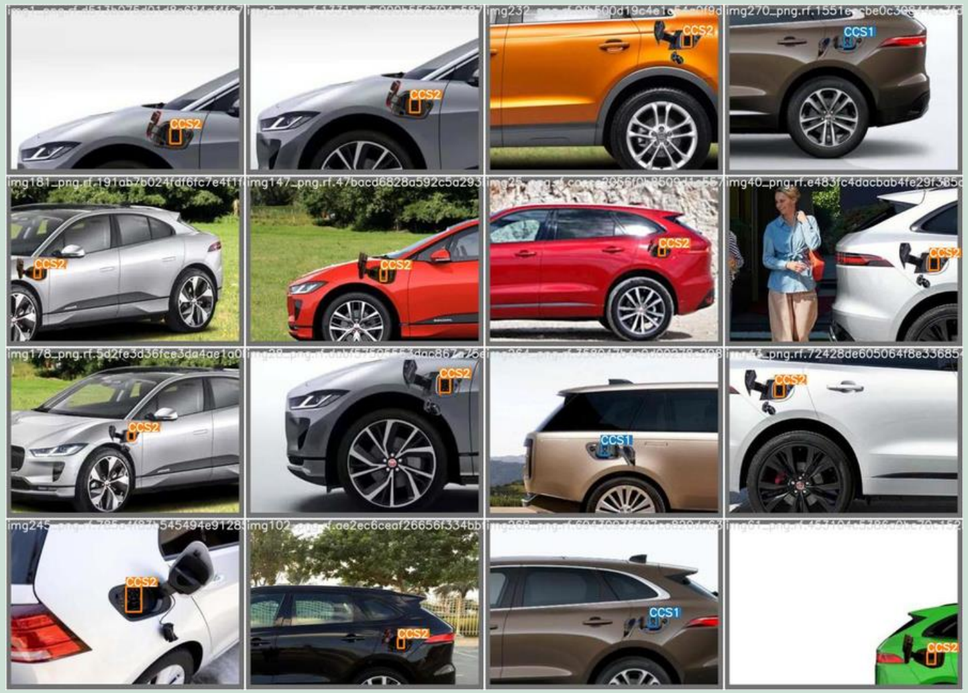
Training Set: Collected hundreds of Jaguar Car images(side view) along with CCS1 and CCS2 charging sockets images and edited their combinations to prepare a dataset of 272 images.

## Raw Dataset:

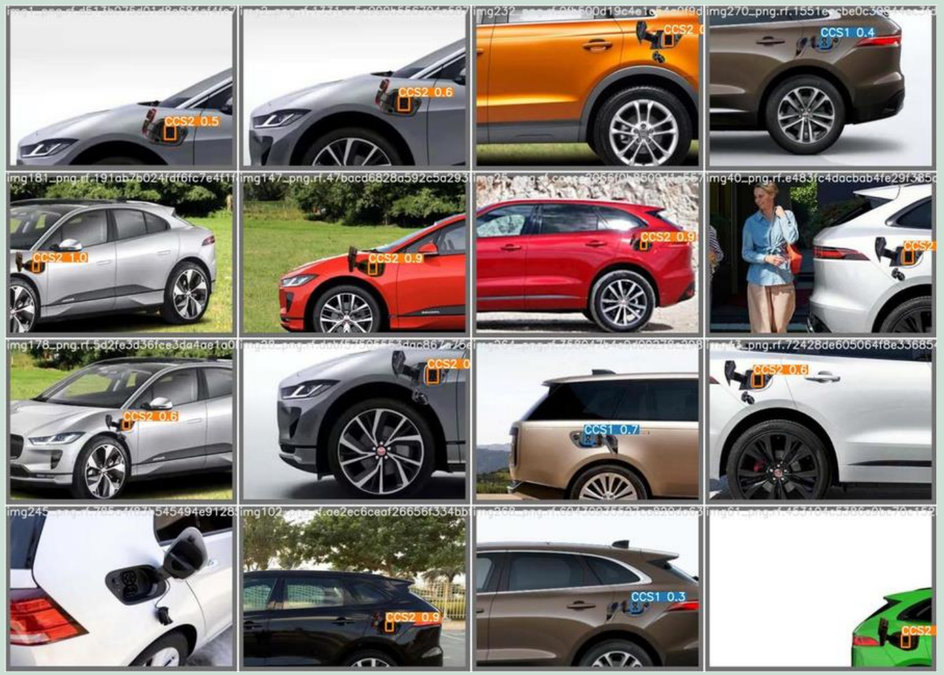



**Dataset Annotation and export were done with the help of roboflow.**

# After training on YOLO v7 for 70 epochs, we got the following results



Test batch-Labels

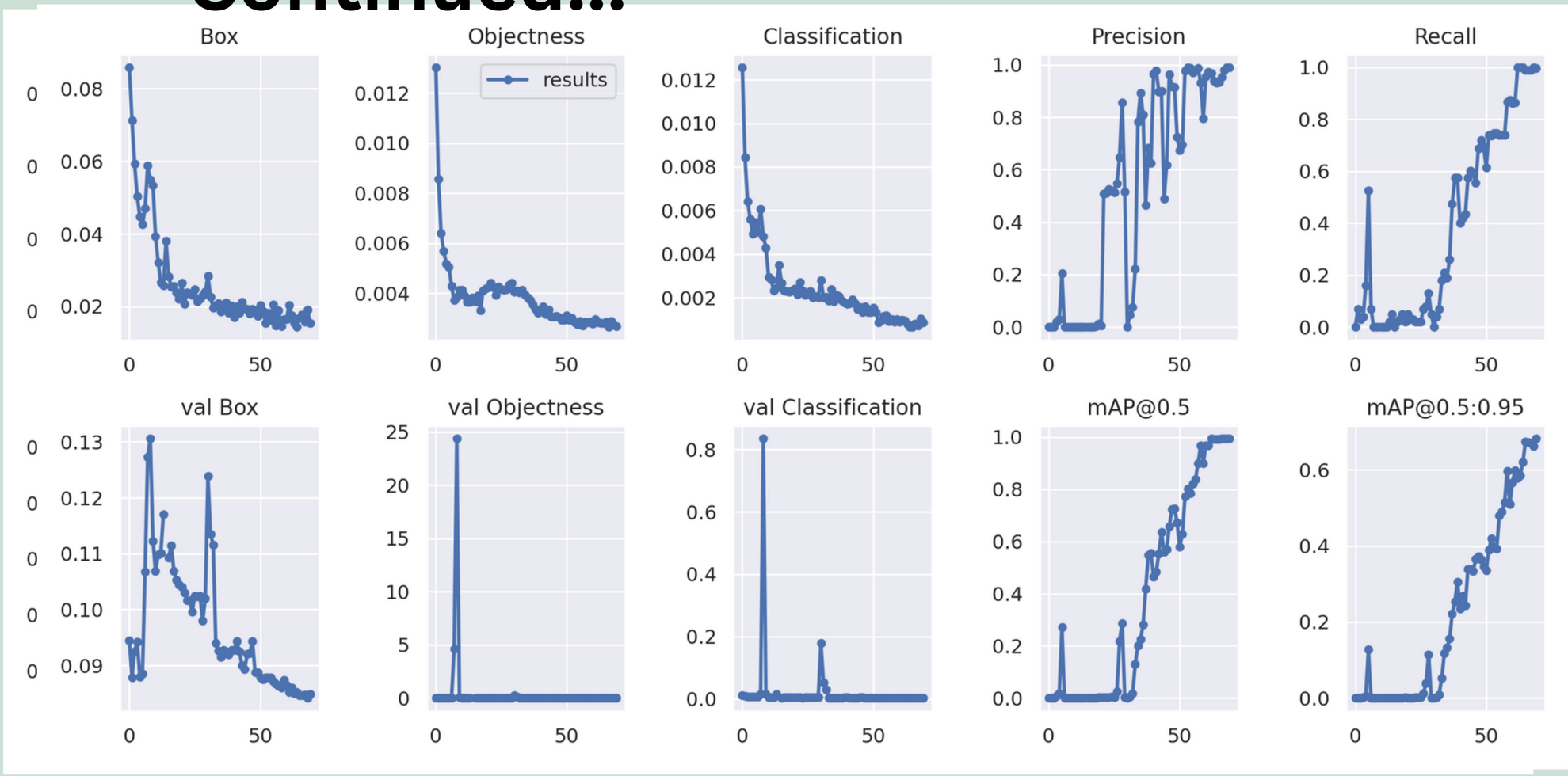


Test batch-Predictions

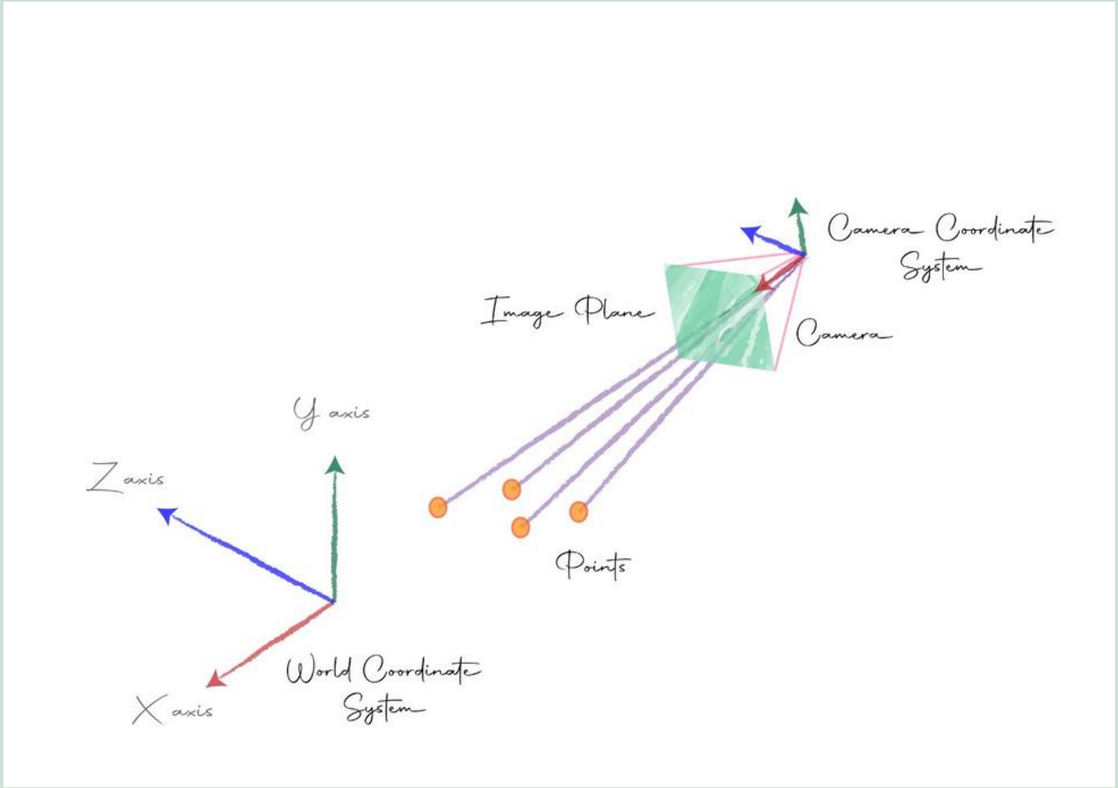
# Results Continued...


# Results Continued...

# Results Continued...



- Now, YOLO v7 returns the coordinates of 4 corners of the bounding box [x1, y1, x2, y2, x3, y3, x4, y4, class]
- We find centre coordinates X = (x1 + x2 + x3 + x4)/4 , Y = (y1 + y2 + y3 + y4)/4
- Then Coordinates transformation is applied on (X, Y) to get back the real (Xreal, Yreal) that the end effector has to reach.



**Now, End effector has reached in front of the socket, we carry forward with the latching procedure.**

**This poses one challenge:**

**The socket may be tilted along any axis (x, y, z)**

# Tackling Challenge

- **Rotated around Z-axis:**

We gave a simple approach to this problem. We created some hundreds of datasets of CCS1 and CCS2 ports rotated along z-axis at different angles and trained it on a CNN Network - VGG-16. We achieved a decent accuracy of 84% on the test set.

| | name | label |
|---|---|---|
| 2 | img1 | 0.5 |
| 3 | img2 | 0.504167 |
| 4 | img3 | 0.507778 |
| 5 | img4 | 0.510556 |
| 6 | img5 | 0.515833 |
| 7 | img6 | 0.5 |
| 8 | img7 | 0.473889 |
| 9 | img8 | 0.514167 |
| 10 | img9 | 0.509444 |
| 11 | img10 | 0.5375 |
| 12 | img11 | 0.448889 |
| 13 | img12 | 0.488611 |
| 14 | img13 | 0.5475 |
| 15 | img14 | 0.465278 |
| 16 | img15 | 0.413056 |
| 17 | img16 | 0.512778 |
| 18 | img17 | 0.5 |
| 19 | img18 | 0.475278 |
| 20 | img19 | 0.571667 |

Annotations

```
        [-0.0100, -0.0213,  0.0087,  ..., -0.0150,  0.0061, -0.0219],
        [-0.0245, -0.0265, -0.0001,  ..., -0.0204, -0.0005, -0.0246],
        [-0.0314, -0.0305, -0.0039,  ..., -0.0199,  0.0031, -0.0281]],
       device='cuda:0')
tensor([[-0.0205, -0.0105, -0.0373,  ..., -0.0550, -0.0211, -0.0172],
        [-0.0256, -0.0116, -0.0387,  ..., -0.0402, -0.0119, -0.0134],
        [-0.0223, -0.0099, -0.0152,  ..., -0.0199, -0.0137, -0.0120],
        [-0.0272, -0.0103, -0.0414,  ..., -0.0557, -0.0171, -0.0134]],
       device='cuda:0')
tensor([[-0.0367, -0.0327, -0.0061,  ..., -0.0229,  0.0084, -0.0288],
        [-0.0304, -0.0294, -0.0036,  ..., -0.0193,  0.0029, -0.0252],
        [-0.0102, -0.0218,  0.0094,  ..., -0.0187, -0.0078, -0.0245],
        [-0.0138, -0.0219,  0.0069,  ..., -0.0158,  0.0076, -0.0243]],
       device='cuda:0')
tensor([[-0.0130, -0.0093, -0.0249,  ..., -0.0179, -0.0154, -0.0114],
        [-0.0261, -0.0118, -0.0338,  ..., -0.0339, -0.0144, -0.0108],
        [-0.0197, -0.0110, -0.0367,  ..., -0.0597, -0.0180, -0.0171],
        [-0.0356, -0.0107, -0.0406,  ..., -0.0458, -0.0170, -0.0149]],
       device='cuda:0')
tensor([[-0.0225, -0.0235,  0.0242,  ..., -0.0325, -0.0295, -0.0326],
        [-0.0148, -0.0228,  0.0065,  ..., -0.0160,  0.0059, -0.0211],
        [-0.0228, -0.0256,  0.0018,  ..., -0.0194, -0.0002, -0.0241],
        [-0.0152, -0.0226,  0.0053,  ..., -0.0171,  0.0077, -0.0238]],
       device='cuda:0')
tensor([[-0.0328, -0.0103, -0.0370,  ..., -0.0471, -0.0159, -0.0131],
        [-0.0228, -0.0086, -0.0412,  ..., -0.0504, -0.0176, -0.0105],
        [-0.0299, -0.0111, -0.0315,  ..., -0.0568, -0.0173, -0.0078]],
       device='cuda:0')
tensor([[-0.0080, -0.0203,  0.0084,  ..., -0.0139,  0.0061, -0.0218],
        [-0.0212, -0.0238,  0.0030,  ..., -0.0191,  0.0054, -0.0251],
        [-0.0092, -0.0208,  0.0090,  ..., -0.0137,  0.0071, -0.0226]],
       device='cuda:0')
Got 73 / 75 with accuracy 97.33 %
```

Training set Accuracy: 97.33%

```
        [-0.0314, -0.0105, -0.0383,  ..., -0.0451, -0.0171, -0.0142],
        [-0.0209, -0.0099, -0.0181,  ..., -0.0244, -0.0137, -0.0136]],
       device='cuda:0')
tensor([[-0.0524, -0.0597, -0.0352,  ..., -0.0364,  0.0923, -0.0323],
        [-0.0100, -0.0204,  0.0094,  ..., -0.0164, -0.0050, -0.0243],
        [-0.0083, -0.0209,  0.0084,  ..., -0.0144,  0.0065, -0.0219],
        [-0.0086, -0.0209,  0.0092,  ..., -0.0144,  0.0061, -0.0220]],
       device='cuda:0')
tensor([[-0.0307, -0.0115, -0.0299,  ..., -0.0305, -0.0122, -0.0104],
        [-0.0233, -0.0086, -0.0357,  ..., -0.0443, -0.0162, -0.0129],
        [-0.0211, -0.0088, -0.0207,  ..., -0.0176, -0.0131, -0.0118],
        [-0.0228, -0.0106, -0.0355,  ..., -0.0376, -0.0161, -0.0075]],
       device='cuda:0')
tensor([[-0.0074, -0.0195,  0.0080,  ..., -0.0140,  0.0038, -0.0214],
        [-0.0095, -0.0208,  0.0092,  ..., -0.0147,  0.0079, -0.0228],
        [-0.0125, -0.0211,  0.0126,  ..., -0.0219, -0.0110, -0.0260],
        [-0.0057, -0.0203,  0.0086,  ..., -0.0137,  0.0009, -0.0222]],
       device='cuda:0')
tensor([[-0.0270, -0.0115, -0.0313,  ..., -0.0296, -0.0121, -0.0113],
        [-0.0197, -0.0099, -0.0316,  ..., -0.0198, -0.0170, -0.0146],
        [-0.0299, -0.0115, -0.0359,  ..., -0.0508, -0.0159, -0.0105],
        [-0.0322, -0.0103, -0.0421,  ..., -0.0604, -0.0164, -0.0154]],
       device='cuda:0')
tensor([[-0.0078, -0.0201,  0.0082,  ..., -0.0142,  0.0038, -0.0214],
        [-0.0116, -0.0225,  0.0112,  ..., -0.0189, -0.0108, -0.0252],
        [-0.0200, -0.0252,  0.0041,  ..., -0.0178,  0.0050, -0.0232],
        [-0.0144, -0.0222,  0.0072,  ..., -0.0158,  0.0070, -0.0235]],
       device='cuda:0')
tensor([[-0.0255, -0.0105, -0.0218,  ..., -0.0192, -0.0132, -0.0095]],
       device='cuda:0')
tensor([[-0.0119, -0.0211,  0.0104,  ..., -0.0181, -0.0032, -0.0241]],
       device='cuda:0')
Got 21 / 25 with accuracy 84.00 %
```

Test set Accuracy: 84%

- **Rotated around X, Y axis:**

A very simple approach to this problem could be using an RGBD camera. But that poses two challenges:

- The cost of production will go high as RGBD cameras are expensive.
- Mounting it on the wire or the end effector would be a challenging task.

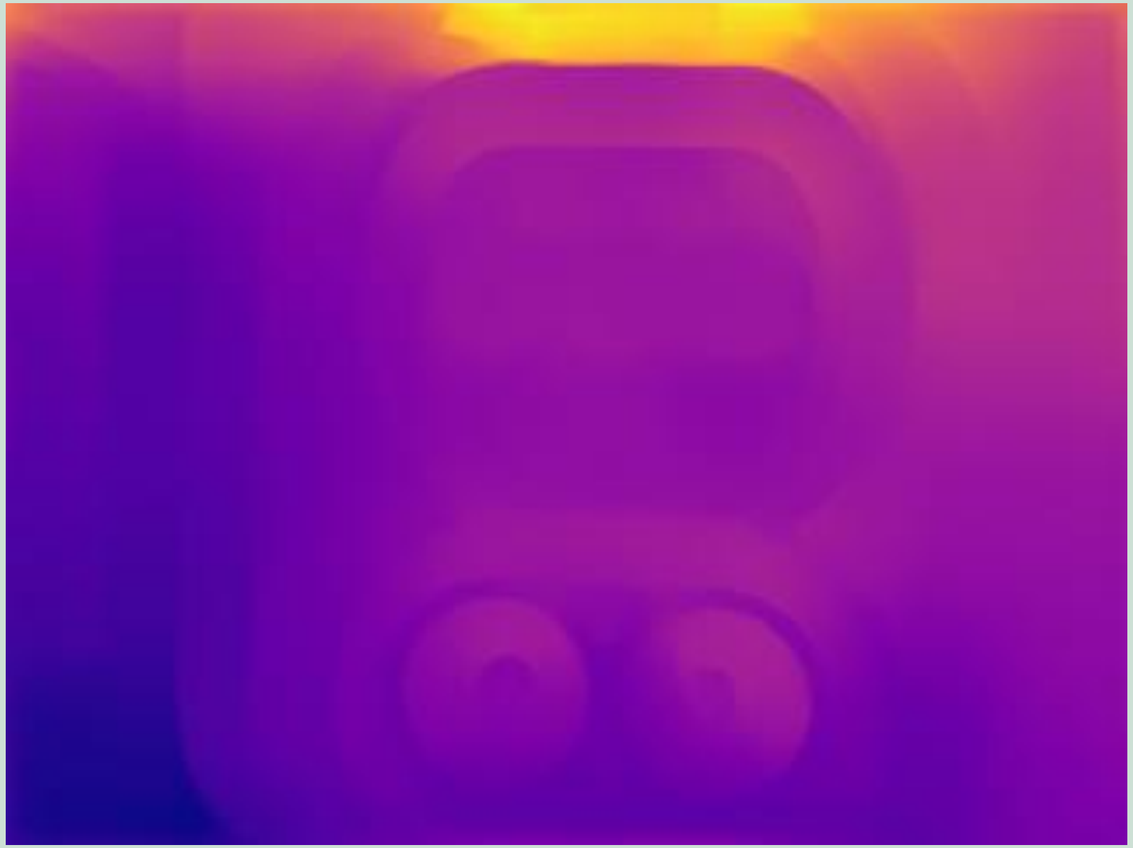
Our Approach: **Monocular Depth Estimation**

It can be implemented with the help of a simple RGB camera.

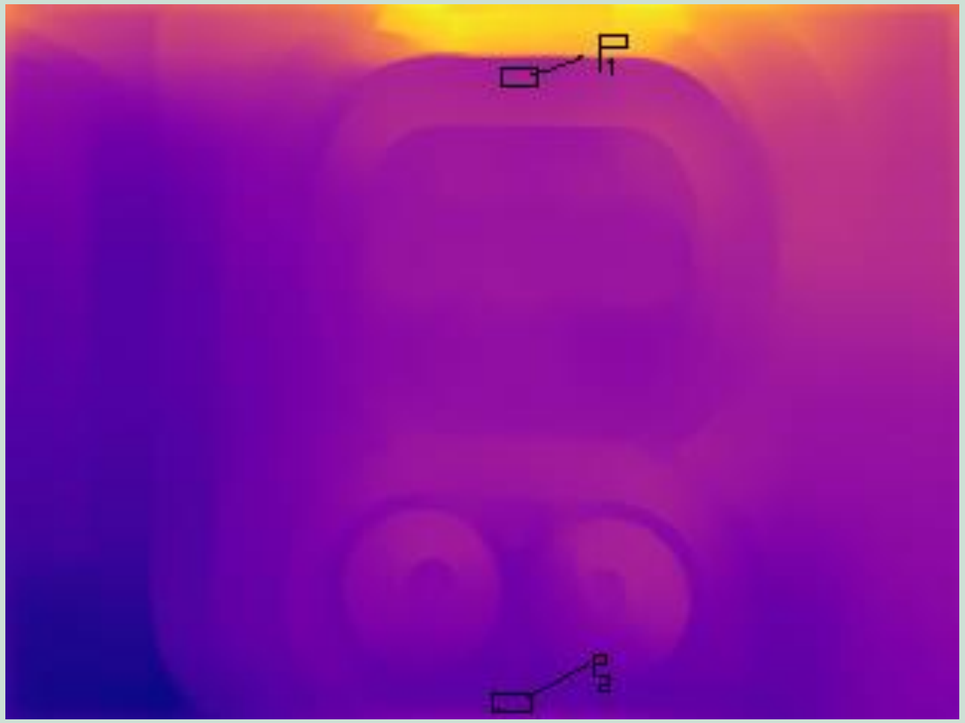
- **Monocular Depth Estimation** is the task of estimating the depth value (distance relative to the camera) of each pixel given a single (monocular) RGB image.

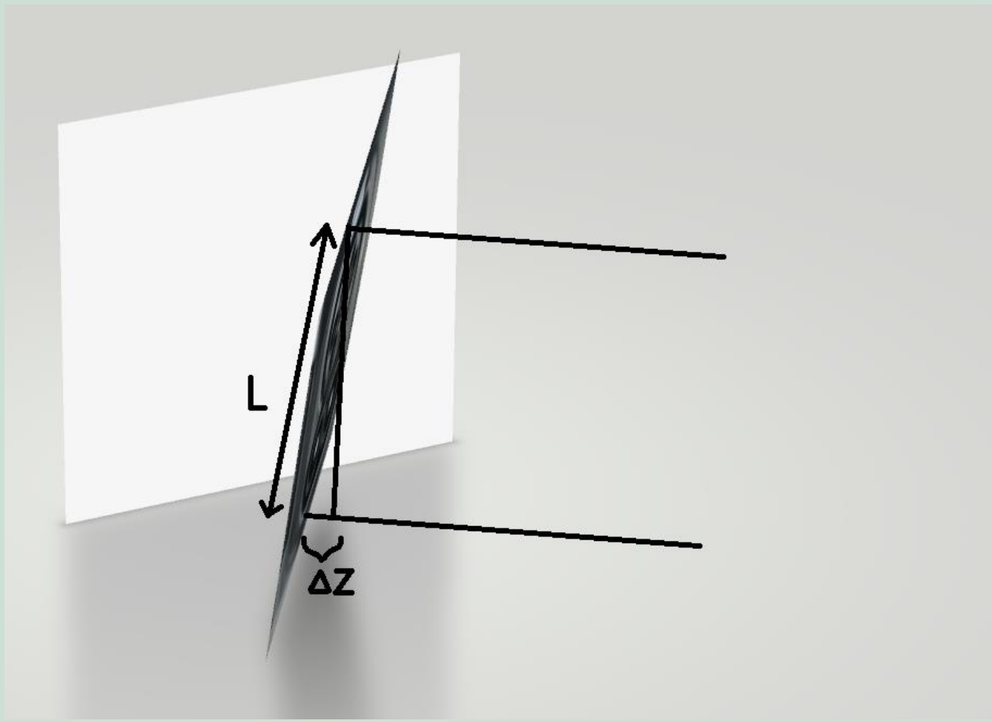
# Applying Monocular Dept Estimation on plugs..

===>



- The **value difference** between the pixels of the top and bottom can be used to calculate Δz which when divided by bounding box length (transformed) will give the angle by which the End effector should rotate along the x-axis.



# Conclusion

Hence, we propose an end-to-end solution to the problem with innovative design and robust vision algorithms.

By integrating all the proposed components, we can achieve autonomous charging for the given workspace range.